A Software Defined Radio Proof-of-Concept Demonstration Platform

Shi Zhong, Craig Dolwin, Rollo Burgess
Toshiba Research Europe Limited, 32 Queen Square, Bristol, BS1 4ND, UK;
{shi.zhong,craig.dolwin}@toshiba-trel.com;
Telephone +44 117 9060700; Fax +44 117 9060701

ABSTRACT

This paper presents our work on a demonstration platform for a software defined radio proof-of-concept. The aim of the demonstrator is to show seamless switching between different functions of an IEEE 802.11a [1] WLAN OFDM system on an adaptive execution environment. The mechanisms and interfaces for dynamic, reliable and secure Software Defined Radio (SDR) equipment reconfiguration are investigated.

The adaptive execution environment concept is evaluated on the Real-Time Research Platform (RTRP). A Functional Description Language (FDL) based on XML is used to describe functional configurations for reconfigurable equipment. The FDL XML description is interpreted by the Configuration Control Module (CCM) using Signal Processing Modules (SPM) to create a binary configuration file for the target platform. A hardware abstraction layer (HAL) for uniform access to the heterogeneous signal processing hardware is defined. Finally a WLAN OFDM system is implemented on the platform and used to demonstrate the ideas discussed.

1. INTRODUCTION

The modern mobile phone works in a heterogeneous radio environment with operators potentially supplying a number of different access technologies. Depending on location and required service the user or operator may select a different technology to minimise cost or optimise performance e.g. when the user is at home or in the office the local WiFi connection is probably the cheapest option for making a connection but when travelling the only option possible maybe a cellular connection. Currently this type of operation is supported by phones that are capable of supporting multiple Radio Access Technologies or waveforms (e.g. GSM, WCDMA, IEEE802.11abg) as well as packet based services such as VOIP. If we project this into the future we can see that, with the gradual push for spectrum de-regulation [2], operators will be free to select their own access technology. This is attractive to the operator because it allows them to differentiate themselves from competitors and rapidly track market requirements. However, it will mean that the number of modes that need to be supported by a multi-mode handset will become very large and the cost and time to develop these multi-mode chips will be prohibitively expensive and slow. The solution we have been investigating in E^{2}R phase 1 and phase 2 is a Software Defined Radio (SDR) that is capable of being dynamically reconfigured while in the field. A basic requirement for any SDR terminal is the ability to be reconfigurable while still maintaining low power consumption and minimal use of silicon. This is a very big challenge to the silicon manufacturer especially when the complexities of future systems are continuing to increase.

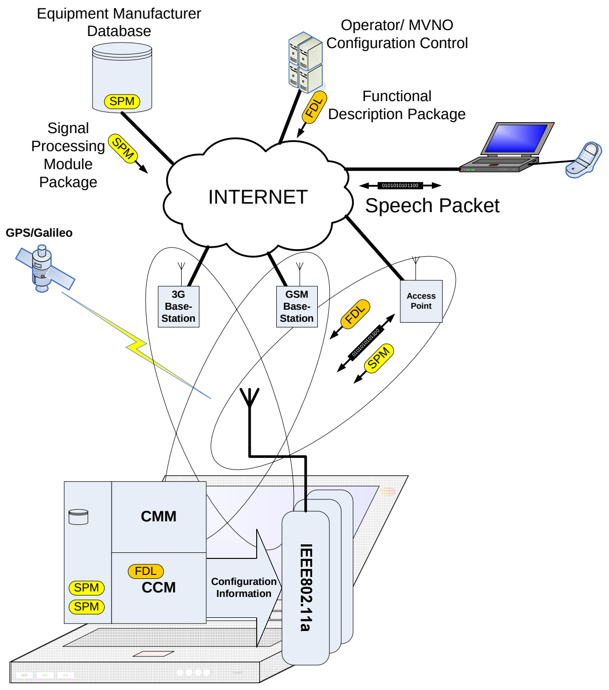
Equipment Manufacturer Database
SPM
Operator/ MVNO Configuration Control
FDL
Functional Description Package
Signal Processing Module Package
SPM
INTERNET
Speech Packet
GPS/Galileo
3G Base-Station
GSM Base-Station
Access Point
FDL
SPM
CMM
FDL
CCM
SPM
SPM
Configuration Information
IEEE802.11a

Figure 1: Reconfiguration scenario

One of the areas E2R has focused on is the logical framework that will allow the remote and reliable reconfiguration of equipment. This has included the definition of a language called Functional Description Language (FDL) to communicate across the network the requirement for a new a configuration. The FDL is platform independent and is interpreted by the equipment, via the

Configuration Control Module (CCM), to create a unique configuration for that specific hardware platform. To create this configuration the CCM may request new object code for reconfigurable logic or DSPs from a manufacturer's database. This is shown graphically in Figure 1. In this paper we describe the work being done within E^2R phase 2 on a proof of concept which aims to demonstrate how these ideas can be implemented on an actual platform.

2. OVERVIEW

The HW/SW block diagram for the SDR Proof-of-Concept demonstration platform is shown in Figure 2.

The Configuration Control Module (CCM) is responsible for the reconfiguration of the radio device. It interacts with a higher-level entity (known as the Configuration Management Module CMM and not shown) which has the responsibility for deciding what Radio Access Technologies (RAT) should be operational at any time. When a change in configuration is required the CMM communicates with the CCM to request new configurations and to exchange any relevant configuration data.

Thereafter a sub-module of the CCM, known as the Primary Configuration Controller (PCC) interprets the configuration request and the configuration data, to decide how best to implement the RAT using the available hardware, ensuring that all relevant constraints are met. Finally the PCC installs the chosen software modules on the signal-processing hardware and initiates the execution of the new RAT. The required implementation-independent functionality of each RAT, including its real-time constraints, is communicated to the PCC by a Functional Description Language (FDL) document.

The Real-time Research Platform [3] (RTRP) contains a set of interconnected hardware components, which are a heterogeneous mix of different processing elements, e.g. GPPs and DSPs, and FPGAs. These provide the mechanisms required for dynamic, reliable and secure change of equipment operation, and offer a consistent interface to the equipment reconfiguration manager in order to apply the needed reconfiguration actions.

A RTRP Hardware Abstraction Layer (RTRP_HAL) has been defined. The RTRP_HAL is a generic interface that allows the client to install tasks, pipes and links in a standard manner. In essence the RTRP_HAL is analogous to an operating system that has been extended to encompass multiple operating systems and a single system-wide communication system that manages the inter-processor links.

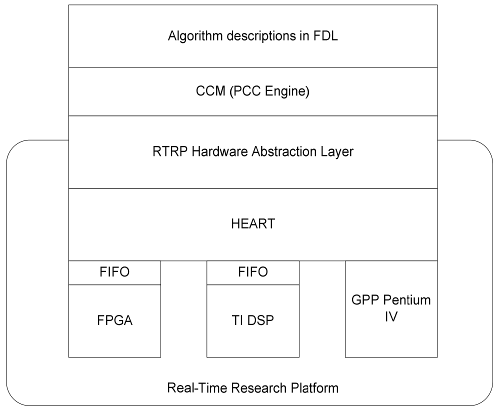
Algorithm descriptions in FDL
CCM (PCC Engine)
RTRP Hardware Abstraction Layer
HEART
FIFO
FIFO
FPGA
TI DSP
GPP Pentium IV
Real-Time Research Platform

Figure 2: Block diagram of the Software Defined Radio Proof-of-Concept Demonstration Platform

3. THE REAL-TIME RESEARCH PLATFORM

The Real-Time Research Platform is a scalable system consisting of a compact PCI rack containing a number of boards, see Figure 3. One contains a dual processor PC running Windows XP while the remainder are carrier boards supporting a flexible communications fabric that links up to 4 processing nodes attached using daughter modules. Both the carrier and daughter boards are commercial off-the-shelf components. The carriers are critical to our approach since they contain communication architecture in the form of a ring that enable nodes to be dynamically connected together, with guaranteed fixed bandwidths and latency.

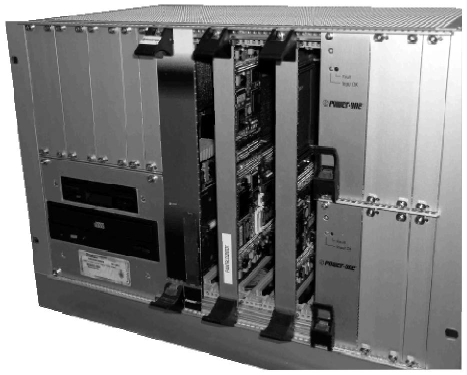

Figure 3: The real-time research platform

Figure 4 shows a typical arrangement of the RTRP with two communication rings and 4 processors. Each module connected to communication fabric can be configured to receive data on one or more of the six available FIFO channels. Similarly each module can write to one or more of six channels. If the total number of channels in a particular HEART [3] configuration is less than six it is possible to allocate more than one time slot to a channel. Each connection between processors across a ring can be assigned bandwidth in increments of 533Mbps up to a limit of 3.2Gbps, the maximum supported by each ring.

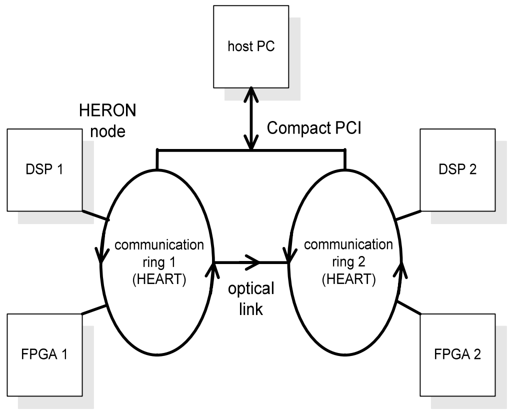

Figure 4: HEART connections of RTRP

The ring's nodes support modules containing processors, ADCs, DACs and inter-ring communication devices. We are using boards containing DSPs and FPGAs.

3.1 DSP module

The DSP module supports a 300 MHz TI C6203 DSP [4] which is connected to both the host system and other modules via the data transfer fabric. The HERON FIFO interface is used to load DSP programs and to allow the DSP to send and receive data from other modules. This module also supports some digital I/O which connects to other system modules.

3.2 FPGA module

The FPGA module provides a Xilinx XC2VP7 Virtex-II Pro [5] FPGA connected to the data transfer fabric. As with the DSP module the FPGA is directly connected to the 100 MHz 32 bit wide HERON FIFO interface. This interface can potentially feed data to the FPGA at gigabit rates whilst simultaneously accepting data at similar rates. The configuration of this interface is programmed via the Hunt host application.

4. FUNCTIONAL DESCRIPTION LANGUAGE

Configuration data is found at all levels of software-defined radio. For example, at the hardware level, a DSP binary file can be considered as configuration data that targets the DSP and configures it for a specific signal-processing function. Likewise an implementation-independent description of that function can be considered as configuration data that targets the configuration plane and is used to configure the software implementation of that part of the radio.

In [6][7][8] we classified the principal configuration data types required for a software modem, showing that many types are well understood, e.g. compiled binary executables for microprocessors. Others, such as radio functional descriptions, are less well known with no commonly accepted representations. A recommendation was made that these types, particularly those concerning high-level descriptions, should be encapsulated using a common data-model, and packaged for transportation and storage using a common file format. Together each data-model and file format constitutes a configuration language. Compressed XML and XML Schema [9] were selected as the file formats for containing the data and data-model respectively.

A functional description language enables radio signal-processing algorithms to be defined as a hierarchical flow of signals (data and control) between functions (termed processes in the language) communicating via 1-to-1 or 1-to-many channels. The channels are connected to input and output ports on the functions. They define the input and output data types. In addition the description captures constraints and timing information, for example a function's real-time deadlines and a channel's maximum latency. The root of a hierarchy is known as an algorithm. An algorithm is just a function which contains other functions and channels. A complete radio modem is an example of an algorithm, one which must be broken up into sub-functions, such as turbo-decoders and filters. There is considerable flexibility in how each algorithm can be implemented, since each sub-function can be targeted to a different processor. Leaf functions, i.e. those that are not algorithms and have no sub-functions, must be implemented using a single target processor.

The first stage in interpreting a functional description is to obtain the individual data items from the XML source. We assume that all such descriptions will initially be available as local XML files. (These may have been obtained either during initialisation of the equipment by a manufacturer/service provider etc, or by over-the-air download.) Subsequently the XML files must be parsed into a memory representation. The data in this representation can either be acted upon immediately or it can be transferred to a more sophisticated storage device, such as a local database. Figure 5 is an overview of the functional description service which is used by the CCM. Since an algorithm is always the starting point the language is known as the algorithm description language.

The language's XML Schema is a critical component; it defines the syntax of the XML documents, the structure of the XML database and classes in the object-oriented implementation of the service. The CCM and this service execute on the host PC of the RTRP. The CCM accesses the service through the algorithm description interface. In general there is a one-to-one mapping between elements in the XML Schema data-model and classes in the C++

implementation. For example the Alg_Algorithm XML element represents an algorithm. The Alg_Algorithm class mirrors the XML element and, being at the top of the hierarchy, provides a large part of the algorithm description interface.

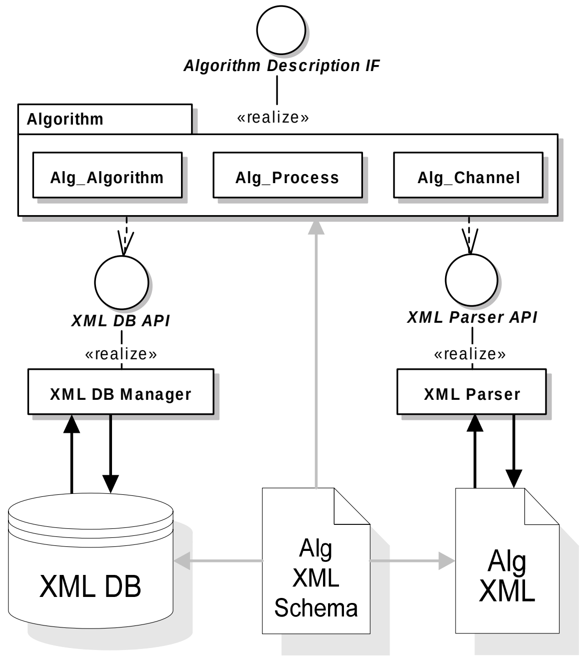

Figure 5: Algorithm description service

5. HARDWARE ABSTRACTION LAYER

The configuration data are parsed by the PCC and implemented on the RTRP platform via RTRP_HAL. In a conventional system access to a hardware device is often achieved using a single operating system, which may be a RTOS if timeliness is an important concern. These operating systems generally apply to a specific type or group of processor types. We assume in the general case the target system will be composed of different types of processors, each potentially with their own operating system. Each OS may support a limited amount of inter-processor communication. In the general case though, as in our hardware architecture, the communication device or devices may be independent of the individual operating systems. From the CCM's perspective, mapping and scheduling the functions is a complex task and it does not want to get involved in the individual details of different hardware. All it needs is to be able to manipulate primitives such as tasks and communication links, with a unified mechanism for reporting errors. The following Table 1 lists some of the generic configuration API for creating tasks, linking two tasks, deleting and abort a task across different execution environment hardware.

TASK_NAME	PARAMETERS
CreateTask	TASK_ID,TASK_NAME, DEST_HARDWARE_ID, BINARY_MEMORY_ADDR, STARTUP_TIME PARAMETERS
DeployTask	TASK_ID
LinkTasks	PRODUCER_TASK_ID, CONSUMER_TASK_ID, MSG_LENGTH
DeleteTask	TASK_ID, WAIT_TO_FINISH
AbortTask	TASK_ID

Table 1. RTRP_HAL configuration API

5.1 Task creation

A task is a program context that can be executed on different hardware. In order to create a task, a unique *TASK_ID* is required. The task instance will be placed into the task database. The *DEST_HARDWARE_ID* specifies the destination hardware, and the *BINARY_MEMORY_ADDR* points to the memory address that stores the binary code for a specific hardware. The *STARTUP_TIME* is used for scheduling purpose, which specifies the time to trigger the task. The *PARAMETERS* provides configuration information for a task.

5.2 Deploy task on a GPP

GPPs provide software programmability by executing the applications on an instruction set architecture (ISA). By changing the software instructions, the functionality of the system is altered without changing the hardware. In order to reconfigure GPP on Windows or Linux operating environment, dynamic loader objects are commonly used, e.g. Windows Dynamic Linkable Libraries (DLL) [10], and Linux shared library (.so) [11]. DLLs need to be relocated during loading, unless the fixed base it was built with happens to be unused in the loading process. Relocations to the same address can be shared, but if different processes have conflicting memory layouts, the loader needs to generate multiple copies of the DLL in memory.

5.3 Deploy task on a DSP

DSPs are specialised processors, which well suited to extremely complex maths-intensive tasks, with conditional processing. The performance of a DSP is limited by the clock rate, and the number of useful operations it can do per clock cycle. TI Dynamic Loader [12] is integrated into the RTRP_HAL to enable tasks to be deployed on the TI 6203 DSP at run-time. The dynamic loader is a general-purpose runtime library for loading program images. It can be used for loading dynamic images to an already running DSP application. When installing a DSP module during execution, the reformatted image is provided to the dynamic loader. The dynamic loader relocates the code and data of the module as needed, and places it in the memory of the TI DSP. References within the module to already loaded software are resolved at this time.

5.4 Deploy Task on a FPGA

FPGAs, contain an array of computational elements whose functionality is determined programmable configuration bits. These elements, sometimes known as logic blocks, are connected using a set of routing resources that are also programmable. In this way, custom digital circuits can be mapped to the reconfigurable hardware by computing the logic functions of the circuit within the logic blocks, and using the configurable routing to connect the blocks together to form the necessary circuit, which in turn provides hardware programmability. In order to deploy a task on VirtexIIpro FPGA chip, the Hunt Serial Bus (HSB) access functions are integrated into the RTRP_HAL. The FPGA can be reconfigured by downloading a new configuration bitstream via HSBs.

5.5 Link creation

Links are created through the RTRP_HAL API, and each link provides one-to-many connection mapping for the execution environment hardware. Figure 6 shows the wrappers for defining link within the hardware, which is suitable for processor type hardware (e.g. GPP, and DSP). When a task is defined, a thread is attached to the task body as a link wrapper. Semaphore and shared memory access functions are also generated by the wrapper. After a result is produced by the producer functions, a semaphore is sent to the consumer thread with the corresponding memory address of the result. Finally, the consumer task will consume the result.

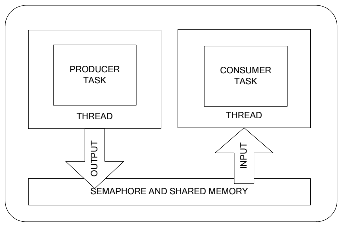

Figure 6: Links between tasks on the same processor

Furthermore, Figure 7 explains a more sophisticated scenario for defining links between tasks on different processor. Extra tasks for crossing hardware communications are needed. In our case, these are Read/Write functions for the communication FIFO between the source and destination hardware. Results generated by a producer task will be routed to the consumer task via the FIFO.

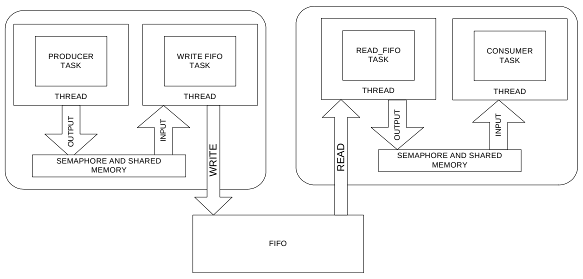

Figure 7: Links between tasks on different processor.

6. WLAN OFDM SYSTEM IMPLEMENTATION

A wireless LAN 802.11a OFDM system is being developed on our proof-of-concept SDR platform. Figure 8 shows the block diagram of the system with a transmit chain and a receive chain. On the transmitter side, the input bits are scrambled, convolutionally encoded, punctured, interleaved, mapped, IFFTed, and added with a cyclic prefix and finally transmitted over RF via an antenna. On the receiver side, the incoming packet is synchronised using the coarse timing block and fine timing block estimates the channel state and information of the packet. The rest of the design blocks of the receiver perform the inverse of the operations performed at the transmitter.

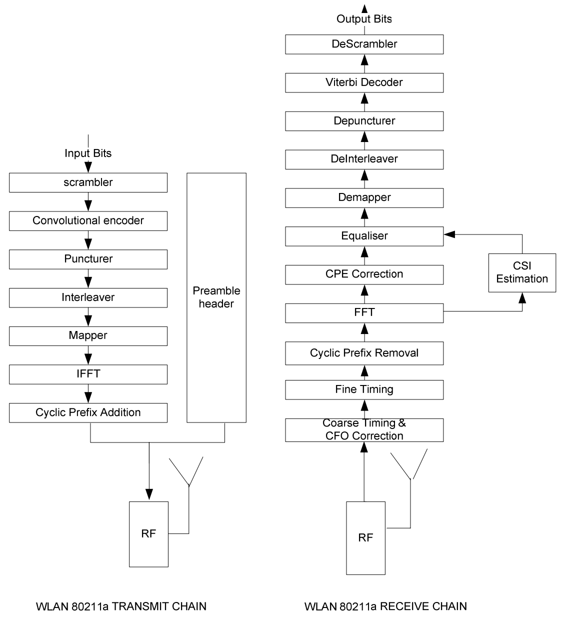

Figure 8: Block diagram of 802.11a OFDM system.

The OFDM system is developed in C. In the preliminary experiment, two common blocks are chosen to demonstrate the switching between FPGA and DSP. Convolutional encoder and 64-point FFT are taken from the OFDM system and compiled on TI Code Composer Studio (CCS) for TI

6203 DSP. The relocatable object codes are loaded into DSP at runtime via the generic configuration API. For the Virtex II-Pro FPGA, convolution encoder and 64-point FFT are generated from Xilinx Core Generator and synthesised by the Xilinx ISE[13] tool flow. Table 2 summarised the size of the 802.11a binary components. For the same implementation, FPGA usually needs more memory than the DSP for storing the configuration data. Compression can be implemented on the configuration data to reduce the memory consumption and latency for downloading such data. FPGA compression works by writing identical configuration frames once rather than many times. Configuration frames are arranged vertically. With the ZIP compression algorithm [14], the bitstream can be compressed much smaller. For example, the size of the compressed convolution encoder bitstream is reduced by 48%, and the zipped version is reduced by 98%.

	DSP object	DSP zipped object	FPGA bitstream	FPGA compressed bitstream	FPGA zipped bitstream
Convolutional Encoder	3K	1K	528K	275K	12K
FFT	13K	3K	528K	376K	77K

Table 2. Size (in bytes) of chosen 802.11a components

Though the FPGA provides a better algorithm implementation with less control flow and more parallel arithmetic and logic operations over DSP, it suffers higher reconfiguration overhead. Table 3 summarises the time to load different components into FPGA and DSP on our RTRP platform. From the table, we see that the FPGA introduces much longer latency for loading the reconfiguration data than the DSP. This is mainly due to the relatively slow serial bus that is used to download bitstreams into the FPGA, and the average achievable bandwidth is less than 100 Kb/Sec. On the other hand, a fast speed FIFO for downloading DSP objects code is used, which guarantees 533Mbps bandwidth. With the compressed configuration data, an extra overhead for decompressing the data is also introduced. But the compression method is regarding as a trade-off between reconfiguration overhead and the time for downloading a component over the air.

	DSP object	DSP zipped object	FPGA bitstream	FPGA compressed bitstream	FPGA zipped bistream
Convolutional Encoder	0.011s	0.058s	2s	3s	3.47s
FFT	0.023s	0.070s	13s	14s	14.47s

Table 3. Loading time of chosen 802.11a components

The size and loading time of the components provide quantised profiling information for the CCM, which can be used to make design-time and run-time decisions for switching components between different execution environment hardware.

7. CONCLUSIONS AND FUTURE WORK

A SDR proof-of-concept demonstration platform is presented in this paper. The hardware architecture of Real-Time Research Platform (RTRP) is introduced. With the FDL description, an algorithm can be interpreted and executed on the platform. RTRP_HAL provides a generic configuration API for loading the components. An IEEE 802.11a OFDM system is being implemented on the platform. Two components are chosen to demonstrate the run-time reconfiguration concept for SDR systems. In the future, a CMM module running on a NOKIA 770 handheld device will be integrated with the current platform, and a mechanism for the dynamic, reliable and secure reconfiguration of SDR equipment will be demonstrated as part of the E^{2}R project.

8. ACKNOWLEDGEMENT

This work has been performed in the framework of the EU funded project E^{2}R II. The authors would like to acknowledge the contributions of their colleagues from E^{2}R II consortium.

9. REFERENCES

[1] IEEE Wireless LAN Edition, A compilation based on IEEE Std 802.11TM-1999 (R2003) and its amendments

[2] OFCOM Spectrum Framework Review Plan, Jan 2005, http://www.ofcom.org.uk/consult/condocs/sfrip/sfip/sfr-plan.pdf

[3] Hunt Engineering website: http://www.hunteng.co.uk

[4] TMS320C600 CPU and Instruction Set Reference Guide, TI, Oct,2000

[5] Virtex-II Pro Platform FPGA Data Sheet, Xilinx, 09/10/03

[6] R. Burgess, S. Mende. The Role of Configuration Data and a Configuration Control Module in an End-to-End (E2R) Software Radio System. 14th IST Mobile & Wireless Communications Summit: Dresden, 19th - 23rd June 2005

[7] R. Burgess, S. Mende. Configuration Languages - Theory and Implementation. E2R Project Whitepaper: http://e2r.motlabs.com/whitepapers

[8] Rollo Burgess, On the Integration of an XML Functional Description Language into the Configuration Controller of a Proof-of-Concept Software Radio System, 4th Karlsruhe Workshop on Software Radios,Germany,22nd & 23rd,2006

[9] W3C, XML website: http://www.w3.org/XML

[10] MSDN, Microsoft online document, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dllproc/base/dynamic_link_libraries.asp

[11] Linux online document, http://www.linux.org/docs/ldp/howto/Program-Library-HOWTO/shared-libraries.html

[12] Getting Started With the Dynamic Loader, TI application Report, December, 2002

[13] Xilinx ISE software Manuals and Help, Xilinx ,2005

[14] Guy E. Blelloch, Introduction to Data Compression, October 16, 2001